BUNDESREPUBLIK DEUTSCHLAND

JC511 U.S. PTO
09/282145
03/31/99

Bescheinigung

Die Siemens Aktiengesellschaft in München/Deutschland hat eine Patentanmeldung unter der Bezeichnung

"Datenhaltungssystem für persistente Daten"

am 29. April 1998 beim Deutschen Patentamt eingereicht.

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

Die Anmeldung hat im Deutschen Patentamt vorläufig das Symbol G 06 F 9/44 der Internationalen Patentklassifikation erhalten.

München, den 1. Oktober 1998
Der Präsident des Deutschen Patentamts
Im Auftrag

Hiebinger

Aktenzeichen: 198 19 205.3

CERTIFIED COPY OF
PRIORITY DOCUMENT

A 8161
6.90
(EDV-L)
01/98

THIS PAGE BLANK (USPTO)

Beschreibung

Datenhaltungssystem für persistente Daten

Die Erfindung betrifft ein Datenhaltungssystem für persistente Daten.

Embedded (gekapselte) Systeme zur Steuerung von elektronischen Geräten müssen eine hohe Zuverlässigkeit aufweisen. Ihre Antwortzeiten auf von außen zugeführte Anweisungen müssen in der Regel sehr kurz sein. Sie unterliegen meist auch strengen Hardwarebeschränkungen. Um die Datenintegrität zu gewährleisten, müssen die Daten stromausfallsicher (persistent) gespeichert werden. Es ist daher erforderlich, entsprechende Datenhaltungssysteme mit permanenten Speichern einzusetzen.

Die Verwendung von Festplattenspeichern ist aus Zuverlässigkeitsgründen wegen ihrer relativ kurzen Lebensdauer und der Empfindlichkeit gegenüber Temperaturschwankungen problematisch.

Aufgabe der Erfindung ist es daher, ein Datenhaltungssystem anzugeben, das die vorstehenden Anforderungen erfüllt und darüber hinaus zuverlässiger und auch kostengünstiger zu realisieren ist.

Diese Aufgabe wird durch das in Anspruch 1 angegebene Datenhaltungssystem gelöst.

Vorteilhafte Weiterbildungen der Erfindung sind in den Unteransprüchen angegeben.

Der wesentliche Vorteil der Erfindung liegt im Zusammenwirken eines schnellen Zwischenspeichers mit einem nur langsam zu beschreibenden permanenten Speicher. Hierdurch können Anweisungen sehr rasch in den Zwischenspeicher übernommen werden

und hierdurch die Antwortzeiten kurz gehalten werden, während unabhängig von diesem Prozeß das Einschreiben von Daten vom Zwischenspeicher in den permanenten Speicher erfolgt.

Vorteilhaft ist die Verwendung von Flash-EPROMS (FEPROM - Flash Eraseable Programmable Memory) als persistenten Speicher, die mit geringem Schaltungsaufwand beschrieben werden können. Die Verwendung von zwei FEPROM-Bausteinen oder zwei Speicherbereichen eines FEPROMS-Speichers , in die abwechselnd sämtliche notwendige Daten eingeschrieben werden, bietet entscheidende Vorteile für einen Neustart des Systems. Bei einem Fehler während des Schreibvorganges - im krassesten Fall durch Stromausfall - steht der bisher gültige komplette Datensatz im anderen Speicherbereich zur Verfügung.

Zur Verringerung der zu speichernden Datenmenge werden nur die für die Konfiguration notwendigen Daten persistent gespeichert.

Das Datenhaltungssystem ist besonders für die Konfiguration von elektronischen Geräten, beispielsweise Datenterminals geeignet.

Im Störungsfall oder bei einem gewollten Wiederanlauf (Restart) werden aus den persistent gespeicherten Daten mit Hilfe von ebenfalls persistent gespeicherten Programmen alle notwendigen Daten der Applikation zur Festlegung der Terminalkonfiguration rekonstruiert - oder bei sehr einfachen Datenhaltungssystemen direkt übernommen.

Vorteilhaft ist auch die Möglichkeit, eine oder mehrere Konfigurationen auf der Datenseite vorzubereiten und diese erst zu einen späteren Zeitpunkt durchzuführen.

Ein Ausführungsbeispiel der Erfindung wird anhand von Figuren näher erläutert.

Es zeigen:

Figur 1 ein elektronisches Terminal mit dem erfindungsgemäßen Datenhaltungssystem,
Figur 2 ein Prinzipschaltbild des Datenhaltungssystem,
Figur 3 ein Prinzipschaltbild zur Erläuterung des Systemneustarts und
Figur 4 ein Hardwarezustandsdiagramm.

In **Figur 1** ist als Beispiel für den Einsatz des erfindungsgemäßen Datenhaltungssystem ein elektronischen Terminals T als Blockschaltbild dargestellt. Bei diesem Terminal handelt es sich um einen synchronen Multiplexer, der über eine erste Datenverbindung L1 und eine zweite Datenverbindung L2 mit anderen Terminals bidirektional Daten austauscht. Über Anschlußbaugruppen (Terminal-Cards) TC1 und TC2 werden diese Daten über ein Koppelfeld CC geführt, das einzelne bidirektionale Datenkanäle DK1 bis DKn aus den externen Datenströmen abzweigt und hinzufügt. Außerdem können die Datenkanäle im Koppelfeld neu geordnet werden, bevor sie vom Terminal weitergesendet werden.

Die Konfiguration (Arbeitsweise) des Koppelfeldes bestimmt beispielsweise, welche Kanäle abgezweigt werden. Dessen Konfiguration und die Funktion der Anschlußbaugruppen werden von einer Systemsteuerung SCU bestimmt, die ihre Anweisungen über eine Datenverbindung DV von einer Zentralsteuerung ZCU erhält. Deren Anweisungen (Befehle) werden von der Systemsteuerung umgesetzt und in konvertierter Form als Steuerinformation SI an die Baugruppen des Terminals weitergegeben, wo die (hardwaremäßige) Konfiguration erfolgt.

Von den Baugruppen des Terminals werden Alarm- und Zustandsmeldungen MI an die Systemsteuerung weitergegeben, die diese umsetzt und zur Zentralsteuerung weiterleitet oder ggf. beispielsweise selbst Ersatzschaltungen vornimmt.

Damit bei einem Stromausfall die vorher eingestellte Konfiguration wieder hergestellt werden kann, müssen alle für die Konfiguration notwendigen Daten, hier als Konfigurationsdaten bezeichnet, in einem persistenten Datenhaltungssystem PDB abgespeichert werden.

In **Figur 2** ist das persistente Datenhaltungssystem PDB als Prinzipschaltbild dargestellt. Alle Konfigurationsdaten sind in einem ersten Speicher ST1, einem Schreib-Lese-Speicher (Random Access Memory) RAM, gespeichert; beispielsweise in Form eines sogenannten Objektmodells. Ein solches Objektmodell beinhaltet sowohl funktionelle als auch hier zur Konfiguration des Terminals dienende persistente Daten (Konfigurationsdaten), die als MIB - Management Information Base - bezeichnet werden.

Bei einer Neukonfiguration, in Figur 2 durch eine Befehlssequenz NKON ausgelöst, werden zunächst die Konfigurationsdaten im ersten Speicher ST1 geändert und in die Steuerinformation SI umgesetzt an die betroffenen Baugruppen gesendet, wo die hardwaremäßige Konfiguration erfolgt.

Das Einschreiben der Daten in den persistenten Speicher erfolgt in mehreren Abschnitten in einem Hintergrundprozeß, dem Speicherprozeß SPR, dessen Wirkungsbereich in Figur 2 angegeben ist.

Das Datenhaltungssystem sorgt dafür, daß die im ersten Speicher ST1 als Objektmodell gespeicherten Konfigurationsdaten gegebenenfalls inklusive Rekonstruktionsdaten als Stream (Datensequenz) unter Verzicht auf vorgegebene feste Speicherbereiche platzsparend in einen Zwischenspeicher ZST eingespeichert werden. Danach kann die durchgeführte Transaktion, beispielsweise eine Neukonfiguration des Terminals, bestätigt werden und eine weitere Transaktion durchgeführt werden.

Der Zwischenspeicher wird aus mindestens zwei funktionell in Serie geschalteten Schreib-Lese-Speichern RAM1 und RAM2 bzw. zwei Speicherbereichen eines RAM-Speichers gebildet, die im folgenden auch als Speicher bezeichnet werden. In den zweiten Schreib-Lese-Speicher werden die in den ersten Schreib-Lese-Speicher eingespeicherten Daten - gesteuert von einer Speicherverwaltung - übernommen, so daß der erste Schreib-Lese-Speicher RAM1 für die Neueinspeicherung von Daten einer weiteren Konfigurationen zur Verfügung steht. Die im zweiten Schreib-Lese-Speicher RAM2 gespeicherten Daten können nun in einen persistenten Speicher PST eingeschrieben werden.

Als persistenter Speicher sind zwei Flash-EPROMS: FEPROM1 und FEPROM2 bzw. zwei Speicherbereiche eines größeren Speichers vorgesehen. Bei kleineren Datenmengen ist es auch denkbar, unterschiedliche Speicherbereiche eines FEPROM-Bausteins zu verwenden.

Das Abspeichern jeweils aller persistenten Daten - MIB - erfolgt alternierend im ersten und zweiten FEPROM bzw. Speicherbereich. Bei größeren Datenmengen ist es sinnvoll, nur geänderte Daten in die betroffenen Segmente (z.B. 64kBytes) des persistenten Speicher zu kopieren.

Zwar erfordert das Einschreiben von Daten in die FEPROM einen relativ großen Zeitaufwand, der jedoch durch die Zwischenspeicherung unkritisch ist, da durch das Kopieren der persistenten Daten MIB in den Zwischenspeicher die unterschiedlichen Prozesse zeitlich entkoppelt sind.

Je mehr „Speicherstufen" der Zwischenspeicher aufweist, desto mehr kurz aufeinanderfolgende Konfigurationen können zwischengespeichert werden und daher bei einem gewollten Neustarten des Systems rekonstruiert werden.
Wenn es während des Einschreibens zu einem Stromausfall kommt, gehen zwar die noch nicht in den Festspeicher PST eingeschriebenen Daten MIB verloren - der bisher die Konfigu-

ration bestimmende Datensatz, die vorhergehende persistent gespeicherte MIB, ist jedoch immer in dem anderen FEPROM vorhanden, so daß eine gültige Konfiguration rekonstruiert werden kann.

Anhand von **Figur 3** soll ein Neustart nach einem Stromausfall näher erläutert werden. Der Neustart erfolgt mit Hilfe eines Startprogrammes STP, das in einem persistenten Programmspeicher PRST gespeichert ist. Durch das Startprogramm wird zunächst eine ebenfalls persistent gespeichert Applikationssoftware inklusive der Datenbanksoftware in den ersten Speicher ST1 geladen(1), (2). Anschließend wird und ein vollständiger Satz persistenter Daten MIB wird aus einem der FEPROMS in den RAM1 geladen(3),(4), um schnellere Zugriffsmöglichkeiten zu haben. Danach erfolgt auf Anforderung (5) programmgesteuert die automatische Rekonstruktion (6) der Konfigurationsdaten in der Applikation durch das Datenhaltungssystem, so daß ein ablauffähiges Programm wiederhergestellt wird.

In den Figuren 2 und 3 sind keine Recheneinheiten dargestellt. Ihre Funktionen sind durch den nur den persistenten Speicher betreffenden Speicherprozeß SPR und die den ersten Speicher umfassenden Applikationsprozesse APR symbolisiert.

Bei einem von der Zentralsteuerung oder Systemsteuerung softwaremäßig ausgelösten neuen Systemhochlauf löscht das Betriebssystem den Inhalt des ersten Speichers ST1. Der Inhalt des Zwischenspeichers bleibt jedoch erhalten, so daß gleich mit der Rekonstruktion aus dem Zwischenspeicher begonnen werden kann.

In sehr einfachen Datenbanken können die Daten des ersten Speichers ST1 auch direkt in den Zwischenspeicher ZST und den persistenten Speicher PST übernommen werden.

Bei unkritischen Zeitbedingungen kann der Zwischenspeicher auch aus nur einem Schreib-Lese-Speicher bestehen.

In **Figur 4** sind die Hardwarezustände einer Baugruppe dargestellt. Oberhalb des waagerechten Striches stimmt der datenmäßig vorgegebene Konfigurationsstand (S) mit dem aktuellen (hardwaremäßig) realisierten Konfigurationsstand (H) überein. So kann bei einer Neukonfiguration (1) der Konfigurationsstand A direkt in den Konfigurationsstand B überführt werden. Der Konfigurationsstand B kann jedoch auch in einem Schritt (2) in den inaktiven Konfigurationsstand übernommen werden. Hier kann der datenmäßige Zustand - mit B bzw. SB (S - Software) bezeichnet - in einem weiten Schritt von B in C und weiter (6) in D geändert werden, wodurch jedoch noch keine Änderung der tatsächlichen (hardwaremäßigen) Konfiguration erfolgt - mit HA, HB, HC bezeichnet. Der Anwender kann im inaktiven Zustand von SC/HB aus entweder in den früheren Konfigurationszustand B (Rückfallkonfiguration) durch Schritt (4) gelangen, oder den neuen Konfigurationszustand C in einem Schritt (5) erreichen.

Entsprechendes gilt für den datenmäßigen Zustand SD, von dem aus wieder (7) die Rückfallkonfiguration B oder in einem Schritt (8) der entsprechende Hardwarezustand HD erreicht werden kann. Die Rückfallkonfiguration wird im inaktiven Konfigurationszustand im Schreib-Lese-Speicher RAM2 gespeichert.

Weitere Einzelheiten der Erfindung sind dem beigefügten Bericht, Seiten 8 bis 22, zu entnehmen.

DBControl

"Ein objektorientiertes Datenhaltungssystem
für Embedded Systeme als C++ Klassenbibliothek
- klein aber mächtig"

Autoren: Petra Brendel
Gerd Schönwolf
Stefan Ziemski

Siemens AG
ÖN TR SW A3
22.01.1998

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

Inhaltsverzeichnis

Seite

1 Einleitung....2

2 Eigenschaften eines Embedded Systems....2

3 Auswahl einer geeigneten Datenbank....3

4 Streammechanismen für Persistenz genutzt....4

5 Clustering von C++-Objekten....4

6 Design des Persistenz-Konzepts....5

6.1 Die Klasse DBManager als Schnittstelle zur Applikation....6

7 Prozeßübergreifende Transaktionssteuerung....8

8 Speichermedien für die Verwaltung der persistenten Daten....9

9 Transaktionen mit Trockenkonfiguration....11

10 Anlaufverhalten der Datenbank....12

11 Layering von DBControl....13

12 Hinweis....14

13 Literaturverzeichnis....14

Siemens AG
ÖN TR SW A3
22.01.1998

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

1 Einleitung

Embedded Systeme gewährleisten in der Regel ohne Eingriff von außen eine sehr hohe Zuverlässigkeit. Ihre Antwortzeiten sind sehr kurz und sie unterliegen strengen Ressourcenbeschränkungen bezüglich RAM und persistenter Speichermedien. Selbst ein temporärer Spannungsausfall kann behandelt werden, da seine Daten stromausfallsicher gespeichert werden. Um die Datenintegrität zu gewährleisten, muß in einem Embedded System ein geeignetes Datenbanksystem eingesetzt werden.

In den folgenden Kapiteln wird das objektorientierte Datenhaltungssystem, DBControl, vorgestellt. DBControl wurde bei der Firma Siemens als Klassenbibliothek in C++ entwickelt. Es realisiert die Persistenz, d.h. die stromausfallsichere Datenhaltung eines Synchronen Multiplexers für die Telekommunikation in Festnetzen. Das Gerät arbeitet mit 32 MB RAM sowie 2 MB Flash-EPROM (FEPROM) als persistentes Speichermedium unter dem Betriebsystem Lynx OS.

Als erstes zeigen wir die Gründe für die Entwicklung dieses Datenhaltungssystems und warum sich eine käufliche Datenbank nicht so geeignet ist. Anschließend erörtern wir das Design von DBControl, der objektorientierten Klassenbibliothek für Persistenz. Der Schwerpunkt liegt bei diesem Thema auf der C++-technischen Realisierung. Darüberhinaus werden wir uns mit der Frage beschäftigen, wie die Daten in ein persistentes Speichermedium, das FEPROM, abgebildet werden. Diese Diskussion umfaßt ein prozeßübergreifendes Transaktionskonzept, das Zeitverhalten von Transaktionen und die Anbindung des FEPROMs. Außerdem werden die bei DBControl eingebauten Sicherheitsmechanismen erläutert. Hier wird das Anlaufverhalten eines Embedded Systems bezüglich seiner Datenbank vorgestellt.

2 Eigenschaften eines Embedded Systems

Die Anforderungen eines Embedded Systems an ein Datenhaltungssystem werden am Beispiel des Synchronen Multiplexers näher beleuchtet.
Der Synchrone Multiplexer hält seine gesamte Konfiguration in einem C++-Objektmodell. Wird der Multiplexer über eine Steuerschnittstelle von außen neu konfiguriert, so wird sein Objektmodell entsprechend geändert. Anschließend gibt er die Daten in konvertierter Form an eine periphere Hardware weiter, die die eigentliche Steuerung übernimmt. Damit der Multiplexer bei einem Stromausfall dieselbe Konfiguration wieder herstellen kann, muß das Objektmodell persistent abgespeichert werden. Dazu ist ein prozeßübergreifender Transaktionsmechanismus notwendig, weil die Objekte über mehrere Prozesse verteilt sind. Als persistentes Speichermedium wurde für den Multiplexer ein FEPROM ausgewählt. Das ist in einem Embedded System robuster und außerdem kostengünstiger als eine Festplatte. Die Zugriffsmechanismen des Datenhaltungssystems müssen an das FEPROM angepaßt sein.

Siemens AG
ÖN TR SW A3
22.01.1998

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

3 Auswahl einer geeigneten Datenbank

Dies hat große Auswirkungen auf die Auswahl eines Datenhaltungssystems. Die Datenbank muß komplett im RAM laufen und darf dort nicht viel Platz in Anspruch nehmen, weil das gesamte Embedded System mit 32 MB RAM für Programm- und Datenbereich auskommen muß. Die Datenbank muß außerdem die Mechanismen zum Beschreiben von FEPROMs unterstützen. Dabei muß das geforderte Zeitverhalten von einer Transaktion pro Sekunde berücksichtigt werden.

Eine käufliche Datenbank bietet einen wesentlich größeren Funktionsumfang, der in Embedded Systemen nicht benötigt wird. Beispielsweise die Unterstützung von verteilten Datenbanken oder die SQL-Schnittstelle sind im Runtime-System der meisten Datenbanken enthalten. Derartige komplexe Konzepte lassen sich auch nicht herausnehmen. Aufgrund der Ressourcenbeschränkung von Embedded Systemen ist hier eine so mächtige Datenbank nicht einsetzbar.

In Abbildung 1 sind die Anforderungen an das Datenhaltungssystem eines Embedded Systems und seine technischen Merkmale zusammengefaßt. Aus diesen Vorgaben ist die C++-Klassenbibliothek DBControl entstanden, die genau die geforderten Features realisiert und dessen Codegröße sehr kompakt ist.

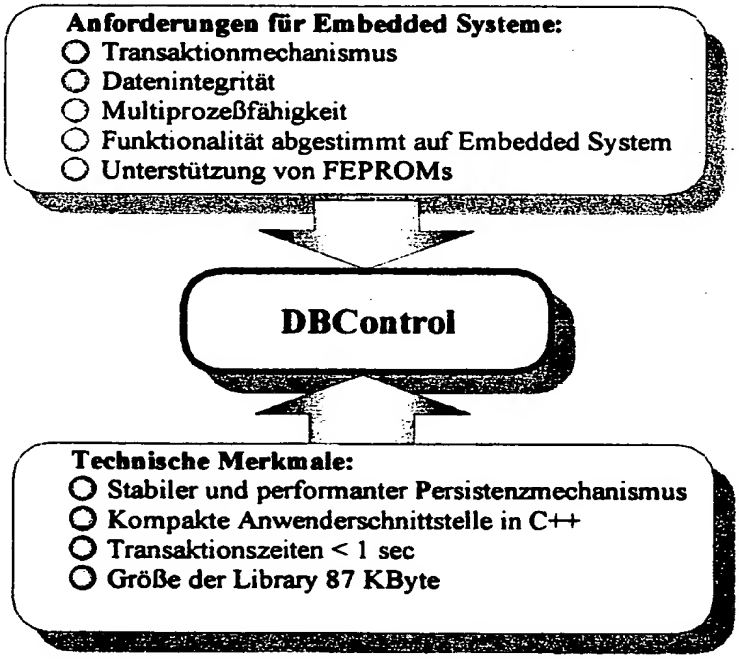


Abbildung 1: Merkmale und Anforderungen des Datenhaltungssystems für Embedded Systeme

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

4 Streammechanismen für Persistenz genutzt

Eine zentrale Aufgabe des Datenhaltungssystems ist die persistente Abspeicherung von C++-Objekten sowie der Beziehungen zwischen ihnen. In DBControl wird diese Aufgabe mit Hilfe von Streammechanismen realisiert. Für die Einbindung dieser Mechanismen benutzt das Datenhaltungssystem die Klassenbibliothek Tools.h++ von Rogue Wave [1]. Hier werden sämtliche persistente Attribute eines Objektes nacheinander in einen bestimmten Speicherbereich kopiert und von dort beim Aufbau der Objekte wieder gelesen.

Tools.h++ bietet zu diesem Zweck die Klasse RWCollectable an. RWCollectable propagiert die virtuellen Methoden saveGuts() für die Abspeicherung und restoreGuts() für das Lesen der persistenten Attribute (siehe Bild 2). Jede Klasse, die persistente Attribute hat, wird von RWCollectable abgeleitet. Sie redefiniert die Methoden saveGuts() und restoreGuts() folgendermaßen:

Die Methoden saveGuts() und restoreGuts() haben einen Stream-Pointer als Parameter. Die persistenten Attribute eines Objektes werden nun mit Hilfe von Output- beziehungsweise Input-Operatoren auf diesen Stream geschrieben oder oder von ihm gelesen. Transiente Attribute werden nicht aufgelistet und dadurch auch nicht abgespeichert.

Mit diesem Mechanismus können Attribute mehrerer Objekte nacheinander in einem Stream geschrieben werden. Um eine C++-Klasse eindeutig zu identifizieren, wird jeweils eine ClassId als eindeutige Nummer mit abgespeichert. Durch sie wird beim Auslesen der Daten entschieden, welches Objekt im Augenblick rekonstruiert wird. Für dieses Objekt wird erst der Default-Constructor und anschließend die restoreGuts()-Methode aufgerufen. Auf diese Weise werden nach und nach alle abgespeicherten Objekte aus einem Stream erzeugt und initialisiert.

5 Clustering von C++-Objekten

Beim Abspeichern gilt die Regel, daß sämtliche Objekte eines Streams neu geschrieben werden müssen, wenn sich ein Objekt innerhalb des Streams ändert. Das liegt daran, daß kein direkter Zugriff auf einzelne Attribute oder Objekte auf dem Stream möglich ist. Alle Attribute werden nacheinander auf dem Stream geschrieben, die genaue Position eines Objektes und dessen Attribute ist aber nicht bekannt.

Um ein ganzes Objektmodell persistent zu speichern und anschließend wieder zu rekonstruieren, genügt es nicht, die Attribute der Objekte abzuspeichern. Beziehungen zwischen den Objekten sind ebenfalls wichtige Informationen, die persistent abgespeichert werden müssen. DBControl unterstützt das Speichern von Pointern zwischen Objekten innerhalb eines Streams. Streamübergreifende Pointer können nicht gespeichert werden.

Daraus ergibt sich folgende einfache Regel für das Clustering der Objekte zu einzelnen Streams: Objekte, zwischen denen persistente Beziehungen existieren, müssen innerhalb eines Streams abgespeichert werden. Andererseits sollten nicht zu viele Objekte einem Stream zugeordnet werden, da sonst das Schreiben des Streams bei Änderung eines Objektes zu lange dauert.

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

6 Design des Persistenz-Konzepts

Abbildung 2 zeigt das Design von DBControl [3]: Die Klasse DBContainer dient als Platzhalter für Streams, indem jeweils ein DBContainer-Objekt genau einen Stream repräsentiert. Die Klasse DBCollectable, abgeleitet von RWCollectable, ist die Vaterklasse aller persistenten Applikationsklassen. Sie vererbt die Methoden saveGuts() und restoreGuts() zum Abspeichern und Initialisieren persistenter Attribute. DBContainer halten einen oder mehrere Pointer auf DBCollectables und bestimmen damit die Zugehörigkeit der Objekte zu genau einem Stream. DBManagerImpl hält mehrere DBContainer-Objekte.

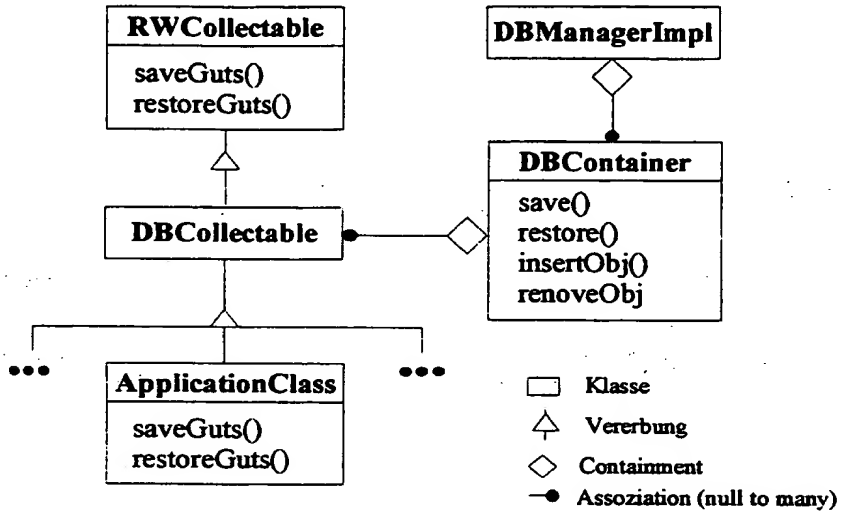


Abbildung 2: Klassendesign von DBControl

Analog dazu können mehrere Streams erzeugt werden. Abbildung 3 zeigt das Layout der Datenbank. Das Layout ist durch die Anzahl und Größe der Streams genau dimensioniert. Die Tabelle DBTable hält sämtliche Verwaltungsinformationen über die einzelnen Streams: Es werden Länge, Anfangspointer und Validflag - d.h. ob der Stream in Verwendung ist - in dieser Tabelle gespeichert.

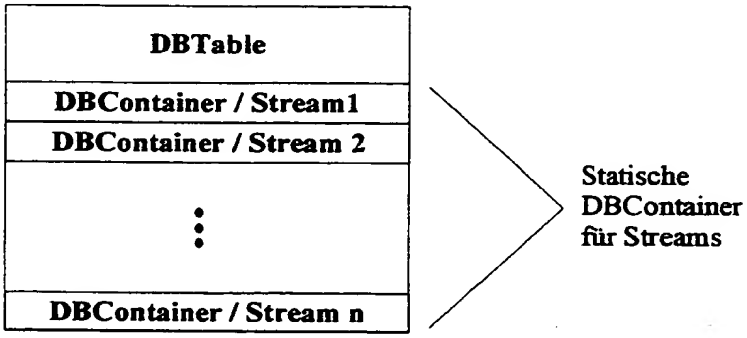


Abbildung 3: Layout der Datenbank

Siemens AG
ÖN TR SW A3
22.01.1998

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

Die Klasse DBManager bildet die Schnittstellenklasse des Datenbanksystems zur Applikation (siehe Abbildung 4). Nach dem Bridge-Konzept von Gamma [2] separiert sie das exportierte Interface von der internen Implementierung, die in DBManagerImpl und den anderen Klassen in Abbildung 2 enthalten ist. Der Vorteil dieses Designkonzepts besteht darin, daß die interne Implementierung in einer Library zusammengefaßt wird. Diese Library kann in der Implementierungs- und Testphase beliebig oft ausgetauscht werden, ohne daß die Applikation deshalb neu übersetzt werden muß. Solange sich die Schnittstelle, d.h. die Headerdatei der Klasse DBManager nicht ändert, muß die Applikation bei einer neuen Version der Library lediglich neu gelinkt werden. Bei Verwendung einer Shared Library ist sogar nur ein Neustart der Applikation notwendig.

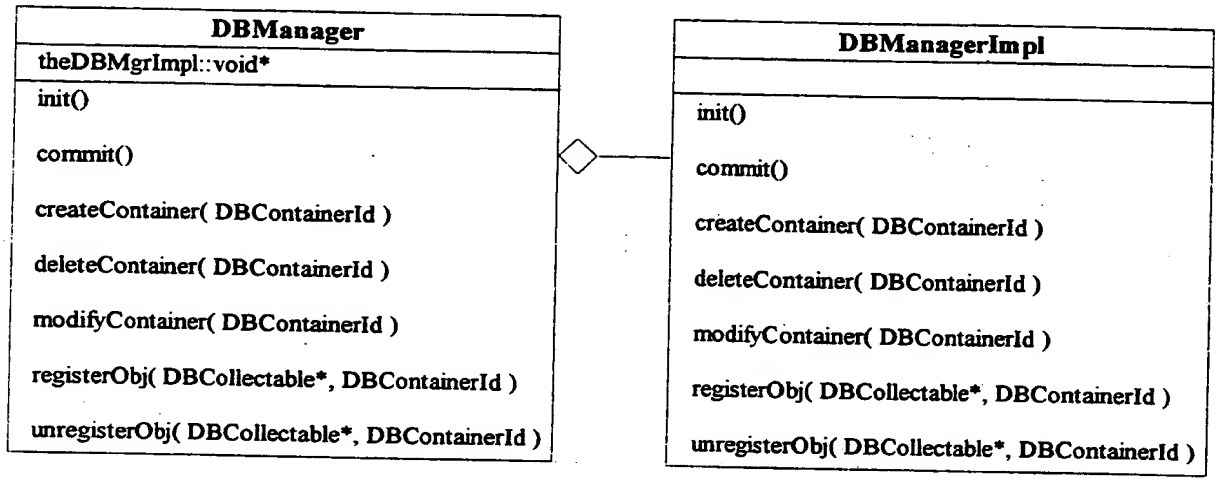


Abbildung 4: Interface von DBControl nach dem Bridge Concept von Gamma

6.1 Die Klasse DBManager als Schnittstelle zur Applikation

In der Klasse DBManager sind sämtliche Funktionen des Datenbank-Systems für den Anwender verfügbar: createContainer() und deleteContainer() dienen dem Erzeugen und Löschen von DBContainern und der zugehörigen Streams. Beim Erzeugen eines Streams wird aus dem Layout (siehe Abbildung 3) ein unbenutzter Stream herausgesucht. Sein Pointer wird in dem DBContainer-Objekt gespeichert. deleteContainer() gibt diesen Speicherbereich wieder frei und löscht das DBContainer-Objekt.

Durch registerObj() wird ein Applikationsobjekt in einem bestimmten DBContainer registriert. Im DBContainer wird die Methode insertObj() aufgerufen und der Pointer auf dieses Objekt wird gespeichert. Ab diesem Zeitpunkt ist dieses Objekt genau dem DBContainer zugeordnet. Durch unregisterObj() wird das Objekt wieder aus dem DBContainer ausgetragen.

Während einer Transaktion markiert die Applikation diejenigen DBContainer durch Aufruf von modifyContainer(), deren Daten sich geändert haben. Am Ende der Transaktion werden

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

die geänderten DBContainer durch Aufruf von commit() in den vorgegebenen Speicherbereich gestreamt. Dabei wird die Methode save() bei der Klasse DBContainer aufgerufen. Sie streamt sämtliche Objekte, die in dem DBContainer registriert sind, durch Aufruf von saveGuts().

Im Gegensatz zum commit() wird beim Aufruf von init() das gesamte Objektmodell aus den Streams ausgelesen und aufgebaut. Es werden nacheinander sämtliche Streams im Datenbank-Layout (siehe Bild 3) auf Gültigkeit überprüft. Enthalten sie gültige Daten, so wird ein DBContainer erzeugt. In der Methode restore() der Klasse DBContainer werden die Applikationsobjekte erzeugt und ihre Attribute mit Hilfe von restoreGuts() aus dem Stream initialisiert.

In Abbildung 5 ist das gesamte Objektmodell, wie es beispielsweise bei der Initialisierung aus den Streams erzeugt wird, zu sehen. Das Objekt DBManagerImpl hält mehrere DBContainer, die wiederum Pointer zu einem oder mehreren persistenten Applikationsobjekten halten. Die Applikationsobjekte eines DBContainers sind teilweise miteinander verpointert. Persistente Beziehungen zwischen Objekten unterschiedlicher DBContainer gibt es nicht.

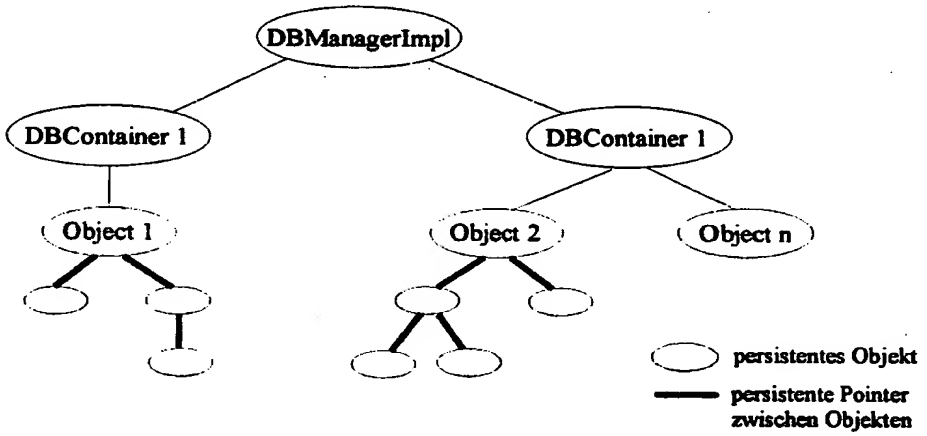


Abbildung 5: Typisches persistentes Objektmodell eines Applikationsprozesses

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

7 Prozeßübergreifende Transaktionssteuerung

Werden durch eine Konfiguration die persistenten Daten einer Applikation verändert, so wird dies in einer sogenannten Transaktion behandelt. Eine Transaktion besteht aus einer Ausführungsphase und der Commit-Phase, in der die Datenbank aktualisiert wird. In einer Embedded Software mit verteilter Prozeßarchitektur muß eine Transaktion und deren Commit prozeßübergreifend behandelt werden. Die Prozesse kommunizieren dabei mittels IPC und SharedMemory.

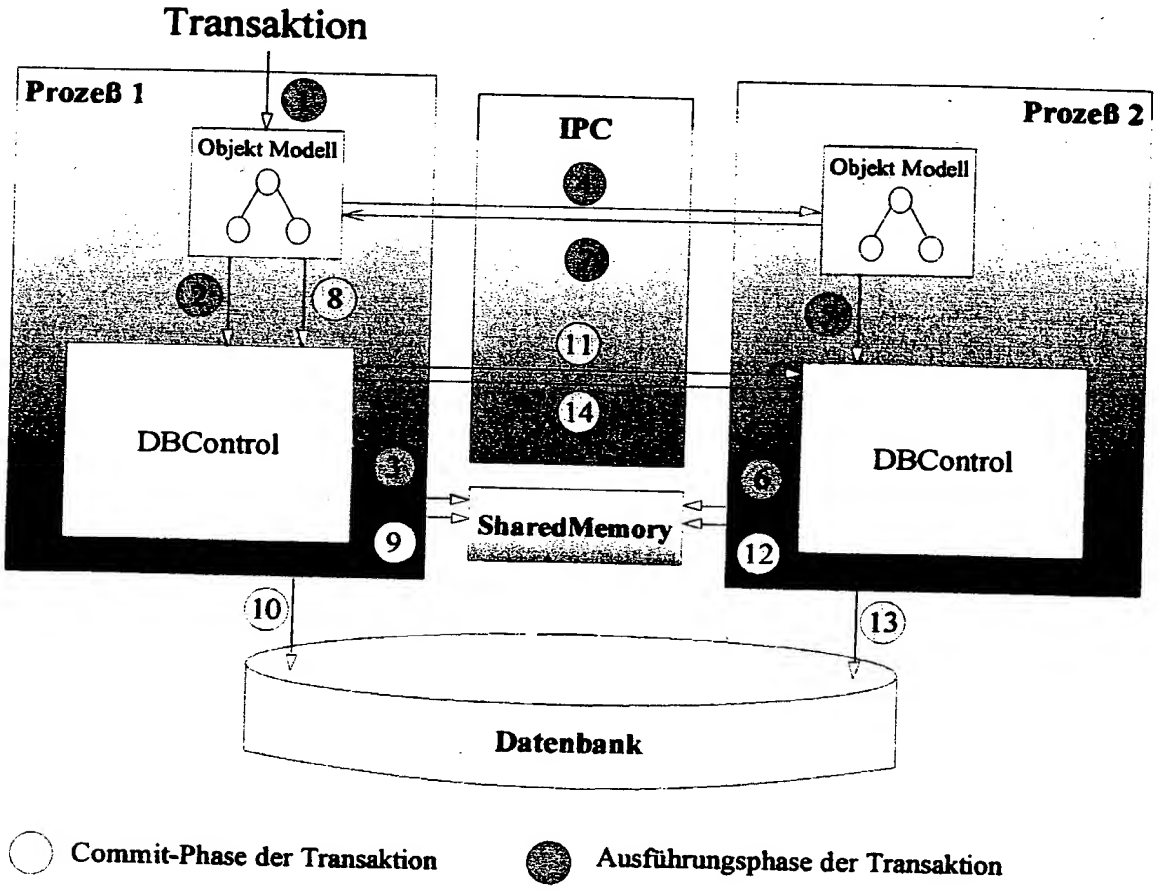


Abbildung 6: Die prozeßübergreifende Transaktionssteuerung in DBControl

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

Abbildung 6 zeigt Schritt für Schritt wie DBControl eine prozeßübergreifende Transaktion steuert. Im Beispiel ist Prozeß 1 für die Transaktion verantwortlich, da sie über ihn getriggert wird (Schritt 1). In der Ausführungsphase (Schritt 2 - 7) werden in beiden Prozessen persistente Daten geändert. Jede diese Änderung wird DBControl über sein Persistenz-Interface mitgeteilt (Schritt 2 und 5). Als Folge davon vermerkt DBControl den Prozeß im SharedMemory mit einen eindeutigen ProzeßIdentifier (Schritt 3 und 6). Am Ende der Ausführungsphase geht die Kontrolle an die Applikation in Prozeß 1 zurück.

Die Commitphase wird durch den transaktionsverantwortlichen Prozeß, im Beispiel Prozeß 1, angestoßen (Schritt 8). Da Prozeß 1 im SharedMemory gekennzeichnet ist (Schritt 9), wird mittels des Persistenzmechanismus die Datenbank aktualisiert (Schritt 10). Der zugehörige ProzeßIdentifier wird aus dem SharedMemory entfernt. DBControl identifiziert im SharedMemory Prozeß 2 als nächsten Kandidaten für den Commit. DBControl in Prozeß 1 schickt DBControl in Prozeß 2 eine Nachricht, die Datenbank zu aktualisieren (Schritt 11). Im Prozeß 2 wird gleichfalls die Datenbank aktualisiert und der ProzeßIdentifier gelöscht (Schritt 12 und 13). Am Ende des Commits wird die Kontrolle an die Applikation des transaktionsverantwortlichen Prozesses, Prozeß 1, zurückgegeben (Schritt 14). Die Datenbank ist konsistent und die Transaktion abgeschlossen.

8 Speichermedien für die Verwaltung der persistenten Daten

DBControl verwendet verschiedene Speichermedien für die Abspeicherung der Datenbank (siehe Abbildung 7). Im RAM wird die Datenbank durch das persistente Objektmodell der Applikationsoftware präsentiert. Dies wird im Falle eines softwaretechnisch ausgelösten Systemshochlauf, im folgenden System Reset genannt, gelöscht. Das pRAM ist ein semipersistentes RAM, das nicht vom Betriebssystem verwaltet wird. Sein Inhalt bleibt bei einem SystemReset unverändert und wird in diesem Fall für die Initialisierung verwendet. Bei einem Stromausfall wird sowohl RAM als pRAM gelöscht und die Datenbank vom FEPROM restauriert.

Damit die Performanz beim Abspeichern der persistenten Daten akzeptabel ist, verwenden wir ein pRAM als Zwischenstation für die Datenbank. Viele Embedded Systeme garantieren eine schnelle Antwortzeit, die beispielsweise eine Transaktionszeit kleiner einer Sekunde erfordern. Eine Transaktion setzt sich aus ihrer Ausführungs- und Commit-Phase zusammen, wobei die Commit-Phase nur einen Bruchteil der gesamten Transaktionszeit in Anspruch nehmen sollte. Dies schließt ein direktes Schreiben der Datenbank ins FEPROM aus, da das Löschen und Beschreiben eines 64-KB-Blocks im FEPROM bereits im Sekunden-Bereich liegt. Das zeitaufwendige Schreiben der Datenbank ins FEPROM muß losgelöst von der Transaktion in einem Hintergrundsprozeß, dem DBServer, durchgeführt werden.

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

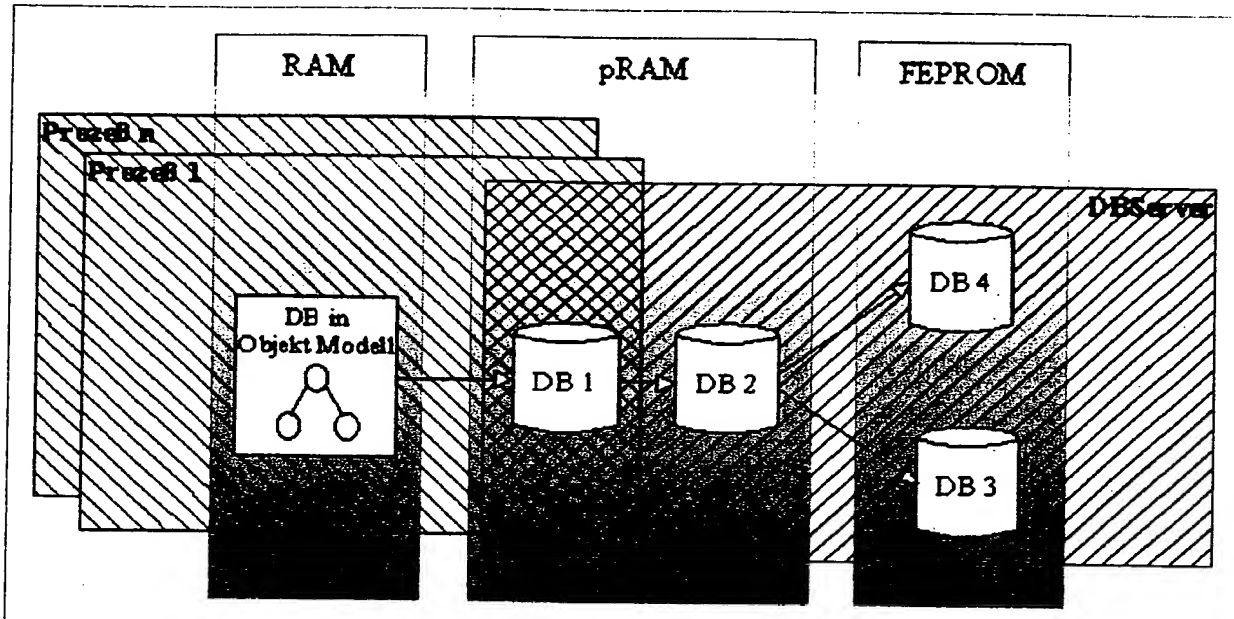


Abbildung 7: Die Datenbankverwaltung über die verschiedenen Speichermedien

DBControl bietet hier folgende Strategie. Die Applikationsprozesse schreiben am Transaktionsende, der Commit-Phase, ihre Änderung im persistenten Objektmodell auf DB 1. Aus Sicht der Applikation ist damit die Transaktion abgeschlossen und die nächste Transaktion kann behandelt werden. Der DBServer-Prozeß wird benachrichtigt, daß eine modifizierte Datenbank in DB 1 vorliegt und kopiert daher DB 1 nach DB 2. Der prozeßübergreifende Zugriff auf DB 1 wird über eine Semaphore gesteuert. Bevor der DBServer das FEPROM beschreiben kann, muß er die entsprechenden Segmente löschen. Durch alternierendes Schreiben auf DB3 und DB4 ist aber auch bei einem Stromausfall eine gültige Datenbank vorhanden. Ist die Datenbank im FEPROM gespeichert, so überprüft der DBServer zyklisch, ob eine weitere Änderung in DB 1 vorliegt.

Liegt die Größe der Datenbank im Megabyte-Bereich, so ist es sinnvoll nur die geänderten Daten zwischen den Datenbanken zu kopieren. Das FEPROM ist in 64 Kbyte Segmente aufgeteilt. Eine intelligente Datenbank-Clusterverwaltung kann das Schreiben auf das FEPROM optimieren, indem nur geänderte Segmente ins FEPROM geschrieben werden.

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

9 Transaktionen mit Trockenkonfiguration

Bei vielen Embedded Systemen lassen sich nicht nur die Konfigurationsdaten persistent abspeichern, sondern zusätzlich eine periphere Hardware einstellen. Für eine bequeme Konfiguration des Gerätes können Transaktionen zunächst ohne Änderung der Hardware durchgeführt werden.

Für diesen Zweck verwaltet die Embedded Software zwei verschiedene Zustände. Im Zustand IDLE wird eine Trockenkonfiguration vorgenommen. Das bedeutet, daß nur die Daten geändert werden, die periphere Hardware bleibt davon unbeeinflußt. Im Zustand ACTIVE wird hingegen bei jeder Änderung auch die Hardware aktualisiert. Es wird deshalb zwischen zwei verschiedenen Konfigurationsständen unterschieden: der Datenstand kann unterschiedlich zur Hardwarekonfiguration sein (siehe Abbildung 8).

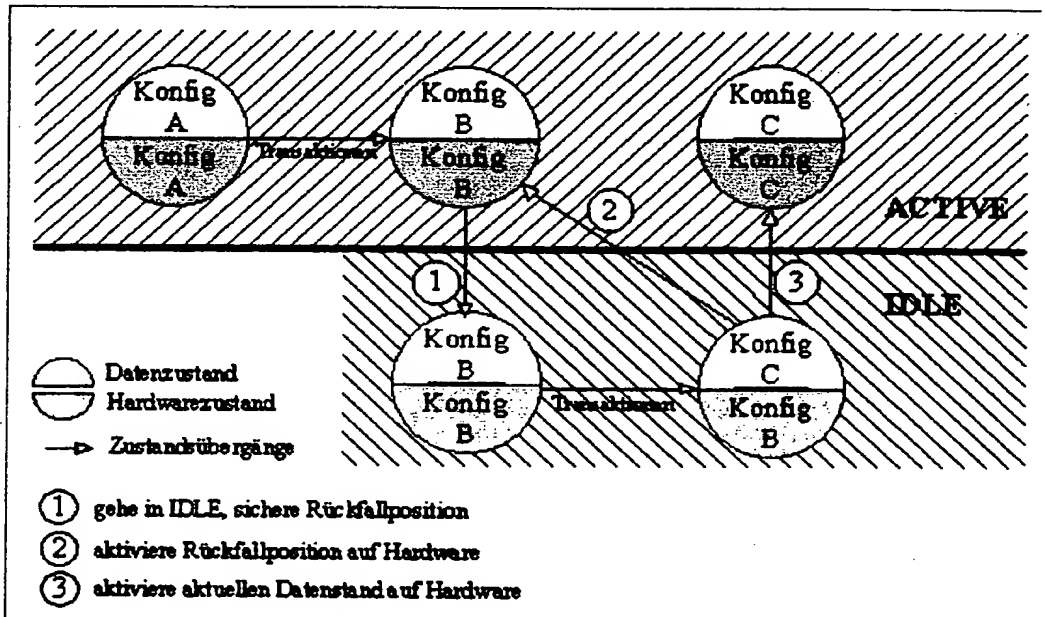


Abbildung 8: Daten- und Hardwarezustandsdiagramm in IDLE und ACTIVE

Abbildung 8 zeigt die Zustandsübergänge zwischen den Zuständen IDLE und ACTIVE. Beim Übergang von ACTIVE nach IDLE wird der aktuelle Datenbestand als Rückfallposition gespeichert (Pfeil 1). Alle folgenden Transaktionen werden ohne Änderung der Hardware durchgeführt. Der Anwender kann danach entweder die geänderten Daten auf der Hardware aktivieren (Pfeil 3) oder wieder die Rückfallposition einnehmen (Pfeil 2). In beiden Fällen geht das Embedded System in den Zustand ACTIVE und die Rückfallposition wird aufgegeben. Die Hardware ist anschließend genauso wie der aktuelle Datensatz konfiguriert.

Dieses Systemverhalten wird auch von dem Datenhaltungskonzept DBControl unterstützt. Im Zustand ACTIVE werden die aktuellen Konfigurationen von DB1 nach DB2 und von dort aus ins FEPROM geschrieben (siehe Abbildung 7). Geht das Embedded System in den Zustand IDLE, so wird die aktuelle Konfiguration im FEPROM als Rückfallposition gespeichert. Der

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

Datenstand im FEPROM wird zu diesem Zeitpunkt das letzte Mal aktualisiert. Erst beim Übergang in den Zustand ACTIVE wird diese Rückfallposition aufgegeben.

Durch diesen Mechanismus hat der Anwender die Möglichkeit, mehrere Transaktionen zusammenzufassen. Erst später wird entschieden, ob diese auch tatsächlich aktiviert werden.

10 Anlaufverhalten der Datenbank

Beim Anlauf eines Embedded Systems muß entschieden werden, welche Datenbank von welchem Speichermedium für den Aufbau des persistenen Objektmodells in den Applikationsprozessen verwendet wird. Das Gerät ist in der Lage, durch Abkopplung zwischen DB1 und FEPROM (siehe Abbildung 7), zwei unterschiedliche Konfigurationsstände zu speichern. Deshalb muß festgelegt werden, welche Datenbank bei einem Systemhochlauf verwendet wird. Außerdem gibt es die Möglichkeit, eine Default-Datenbank zu aktivieren. Sie beinhaltet die Initialwerte des persistenten Objektmodells, das von den Applikationsprozessen aufgebaut werden kann.

DBControl realisiert das Anlaufverhalten folgendermaßen: Grundsätzlich wird bei einem Stromausfall immer aus dem FEPROM gelesen. Bei einem System Reset wird anhand des Zustands des Geräts entschieden, ob aus dem pRAM oder aus dem FEPROM gelesen wird. Ist das Gerät im Zustand IDLE, so wird die Rückfallposition im FEPROM als Datensatz verwendet. Im Zustand ACTIVE wird DB1 aus dem pRAM gelesen, falls hier gerade ein gültiger Datensatz gespeichert ist. Möglicherweise ist der System Reset während dem commit() ausgelöst worden und DB1 beinhaltet einen inkonsistenten Datensatz, da er nicht komplett geschrieben wurde. In diesem Fall wird auf das FEPROM zurückgegriffen. Die Gültigkeit eines Datensatzes wird durch ein Valid-Flag gekennzeichnet.

Die Komponente Anlauf von DBControl (siehe Abbildung 8) kopiert nach diesen Kriterien den aktuell gültigen Datensatz in DB1. Die Applikationsprozesse initialisieren anschließend aus DB1 heraus ihre DBContainer und persistenten C++-Objekte. Bei der Rekonstruktion der Daten wird auch der Zustand IDLE oder ACTIVE aus dem Datensatz gelesen. Abhängig von diesem Zustand wird beim Hochlauf die periphere Hardware mit dem aktuellen Konfigurationsstand initialisiert. In IDLE behält die Hardware ihre Konfiguration getrennt vom Datensatz des Embedded Systems. Soll der Default-Datensatz aufgebaut werden, so werden sämtliche DBContainer in DB1 von der Komponente Anlauf ungültig gesetzt. Die Applikationsprozesse müssen dann selbständig ihr Default-Objektmodell aufbauen.

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

11 Layering von DBControl

Das Datenhaltungssystem DBControl wird von der Applikation als "SimpleService" Schicht benutzt. Die Abbildung 3 zeigt das daraus resultierende Layering der Embbeded Software.

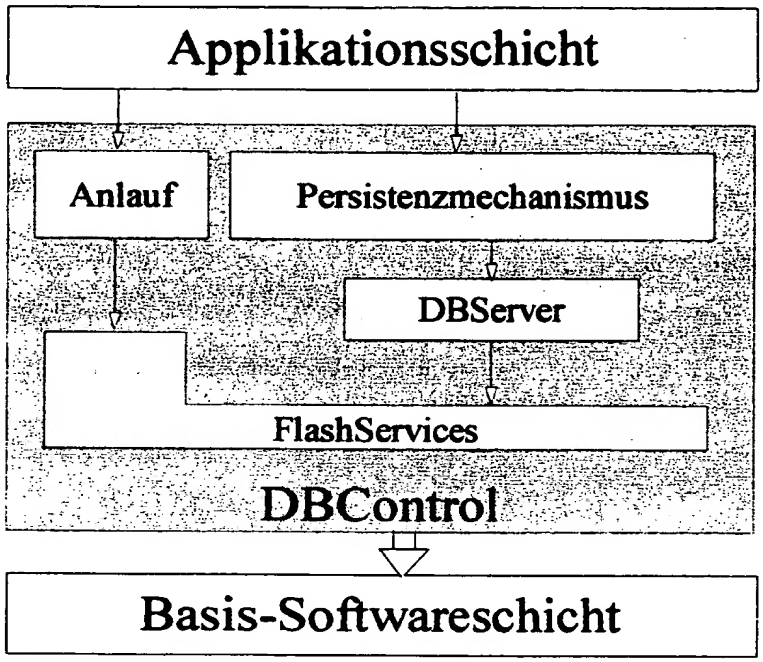


Abbildung 9: Das Schichtenmodell der Applikation aus Sicht von DBControl

DBControl läßt sich in mehrere aufeinander aufbauende Komponenten aufteilen. Der Persistenzmechanismus wird in der laufenden Applikation für die Abspeicherung der persistenten Daten in DB 1 verwendet. Der DBServer ist ein eigenständiger Prozeß, der für die asynchrone Speicherung der Datenbank ins FEPROM über DB2 verantwortlich ist. Die FlashServices umfassen die Treiber für das FEPROM, die die Lese- und Schreibzugriffe realisieren. Wird beim Anlauf eines Embedded Systems die Datenbank aus dem FEPROM restauriert, so benutzt DBControl wieder die FlashServices.

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

12 Hinweis

Zusätzliche Beschreibungen für DBControl liegen in Form einer Funktionsspezifikation, einer Designspezifikation und dem kompletten Sourcecode vor.

13 Literaturverzeichnis

[1] Tools.h++ Introduction and Reference Manual, Version 7,
Rogue Wave Software Inc.

[2] Design patterns. Elements of reusable object-orriented software,
E. Gamma, R. Helm, R. Johnson, Addison Wesley Verlag, 1997

[3] Obect-Oriented Modeling And Design,
James Rumbaugh, Prentice Hall 1991

Petra Brendel, Petra.Brendel@oen.siemens.de
Gerd Schönwolf, Gerd.Schönwolf@oen.siemens.de
Stefan Ziemski, Stefan.Ziemski@oen.siemens.de

Patentansprüche

1. Datenhaltungssystem für persistente Daten
mit einem Zwischenspeicher (ZST; RAM1, RAM2), in den sämtliche persistent zu speichernde Daten (MIB) eingeschrieben werden, und
mit einem an den Zwischenspeicher(ZST; RAM1, RAM2) angeschalteten persistenten Speicher (PST; FEPROM1, FEPROM2), der zwei Speichereinheiten oder Speicherbereiche (FEPROM1 und FEPROM2) aufweist, in denen jeweils die gesamten persistenten Daten (MIB) aus dem Zwischenspeichers (ZST; RAM1, RAM2) gespeichert werden.

2. Datenhaltungssystem nach Anspruch 2,
d a d u r c h g e k e n n z e i c h n e t ,
daß die gesamten im Zwischenspeicher (RAM1, RAM2) gespeicherten persistenten Daten (MIB) abwechselnd jeweils in eine der Speichereinheiten oder Speicherbereiche (FEPROM1 oder FEPROM2) des persistenten Speichers (PST) eingeschrieben werden.

3. Datenhaltungssystem nach Anspruch 2,
d a d u r c h g e k e n n z e i c h n e t ,
daß nur geänderte Datenfolgen abwechselnd in betroffene Speichersegmente des persistenten Speichers (PST) eingeschrieben werden.

4. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
d a d u r c h g e k e n n z e i c h n e t ,
daß in den Zwischenspeicher (ZST; RAM1, RAM2) nur die persistenten Daten (MIB) gegebenenfalls inklusive Rekonstruktionsdaten aus einem ersten Speicher (ST1) übernommen werden, der ein Ablaufprogramm und die zugehörigen persistenten Daten beinhaltet.

5. Datenhaltungssystem nach Anspruch 5,

dadurch gekennzeichnet,
daß die Daten (MIB) im Zwischenspeicher (ZST; RAM1, RAM2) und im permanenten Speicher (PST) platzsparend als Datensequenz gespeichert werden.

6. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
dadurch gekennzeichnet,
daß mindestens ein weiterer permanenter Speicher (PRST) für ein Startprogramm und Applikationssoftware einschließlich Datenbanksoftware vorgesehen ist, mit deren Hilfe aus den im permanenten Speicher (PST; EPROM1, EPROM2) gespeicherten persistenten Daten (MIB) automatisch die in den ersten Speicher (ST1) einzuschreibenden Konfigurationsdaten rekonstruiert werden.

7. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
dadurch gekennzeichnet,
daß es zur persistenten Konfiguration von Funktionen und/oder Eigenschaften eines Terminals (T) und insbesondere von dessen Anschlußbaugruppen (TC1, TC2, ...) vorgesehen ist.

8. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
dadurch gekennzeichnet,
daß als Zwischenspeicher (ZST) mindestens zwei funktionell in Serie geschaltete Schreib-Lese-Speicher (RAM1, RAM2) vorgesehen sind, wobei die im ersten gespeicherten persistenten Daten (MIB) in den zweiten werden, so daß der erste Schreib-Lese-Speicher (RAM1) zur Neueinspeicherung zur Verfügung steht, während die persistenten Daten (MIB) aus dem zweiten oder einem weiteren Schreib-Lese-Speicher (RAM2) in den permanenten Speicher (PST) eingeschrieben werden.

9. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,

d a d u r c h g e k e n n z e i c h n e t ,
daß als permanenter Speicher beschreibbare Flash Eraseable Progammable Read Only Memory-Bausteine vorgesehen sind.

10. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
d a d u r c h g e k e n n z e i c h n e t ,
daß mehrere Konfigurationen gespeichert werden und eine dieser Konfigurationen für die harwaremäßige Realisierung auswählbar ist.

11. Datenhaltungssystem nach einem der vorhergehenden Ansprüche,
d a d u r c h g e k e n n z e i c h n e t ,
daß zunächst mehre Konfigurationsänderungen nur auf der Datenhaltungsseite durchgeführt werden und
daß anschließend eine alle Konfigurationsänderungen umfassende funktionsmäßige und/oder hardwaremäßige Änderung im Terminal (T) durchgeführt wird.

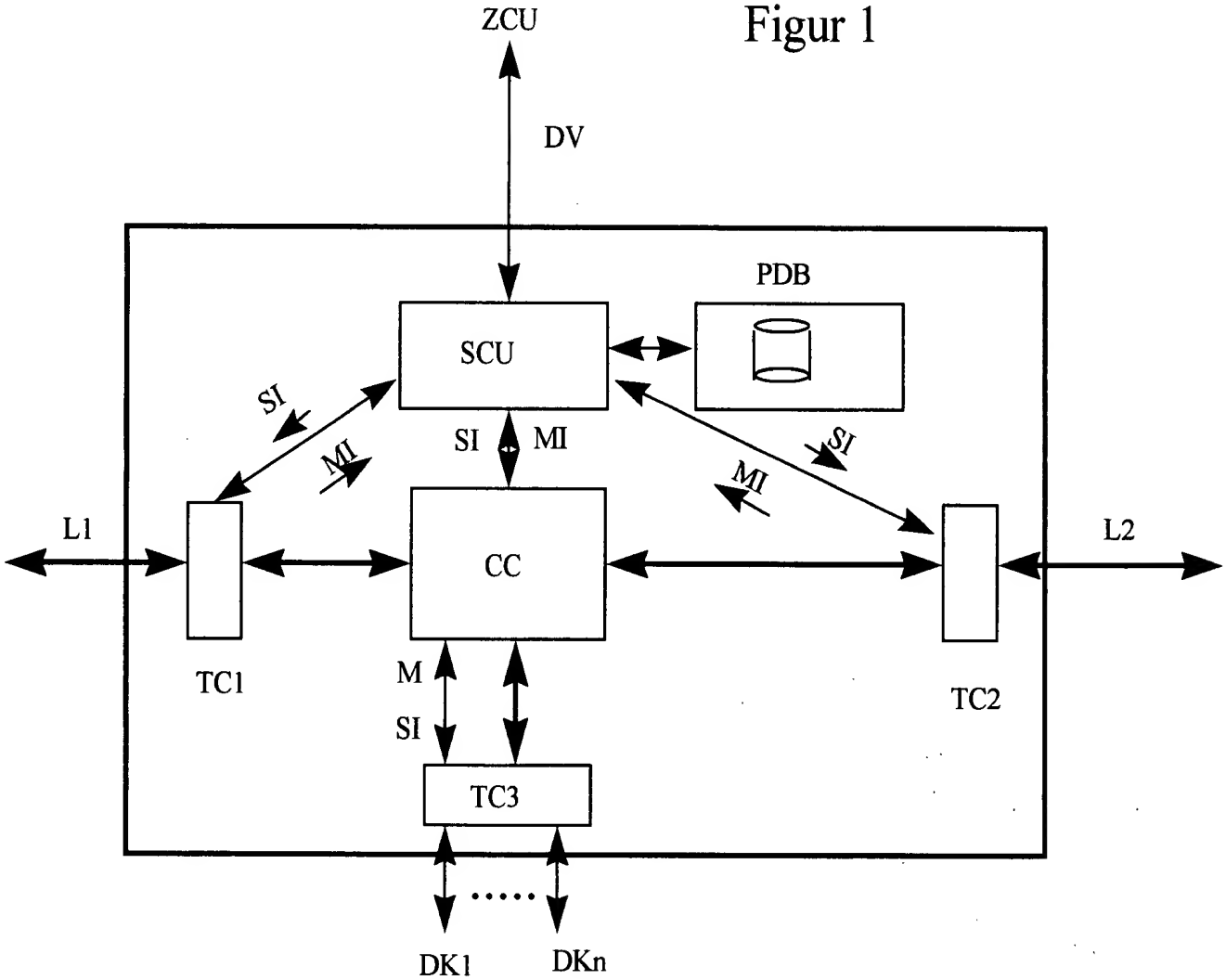
Zusammenfassung

Datenhaltungssystem für persistente Daten

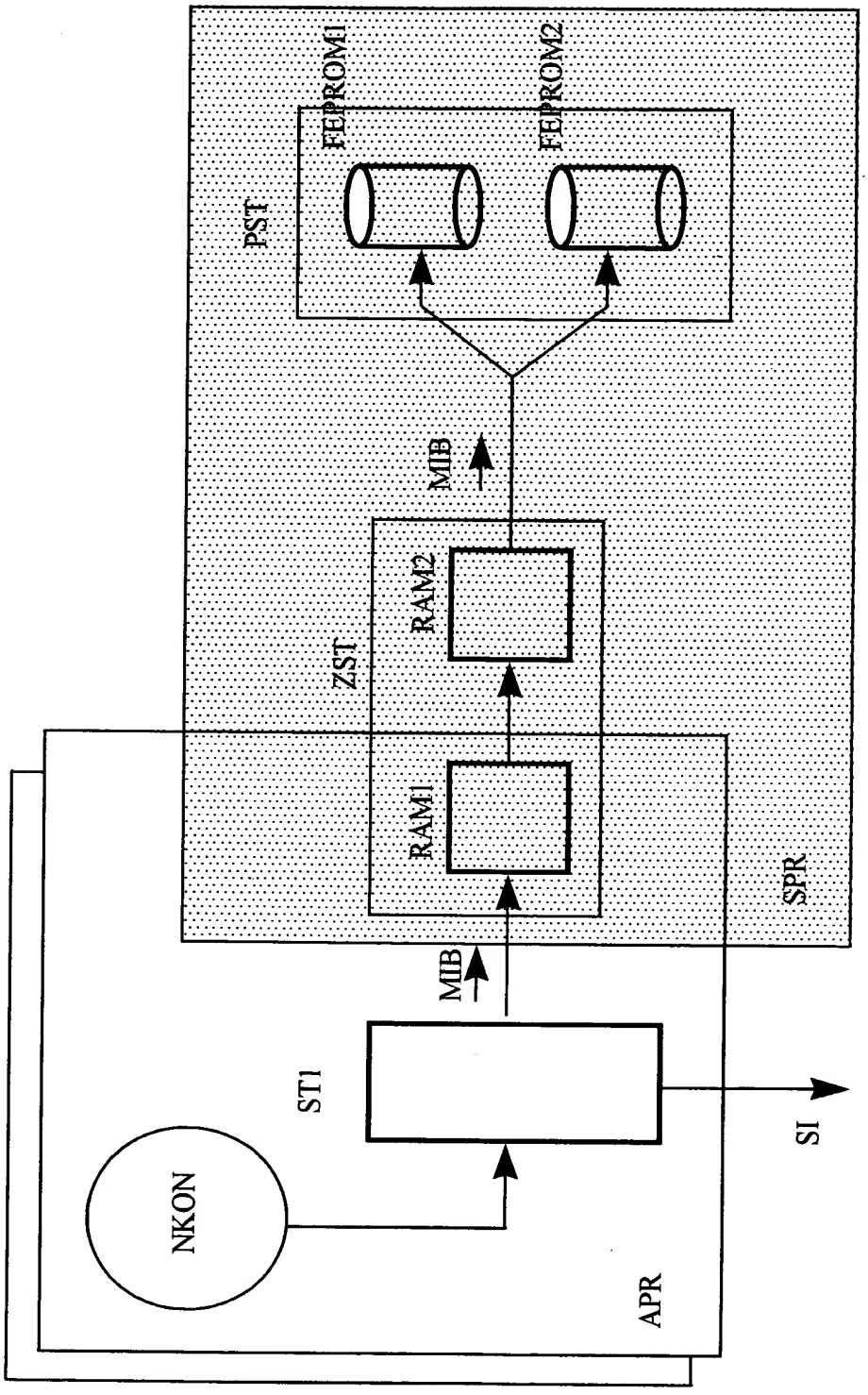
Datenhaltungssystem für persistente Daten mit einem ersten Speicher, aus dem Daten in einen Zwischenspeicher (ZST; RAM1, RAM2) eingeschrieben werden, und mit zwei persistenten Speichern (PST; FEPROM1, FEPROM2), in die Daten aus dem Zwischenspeicher (ZST) übernommen werden.

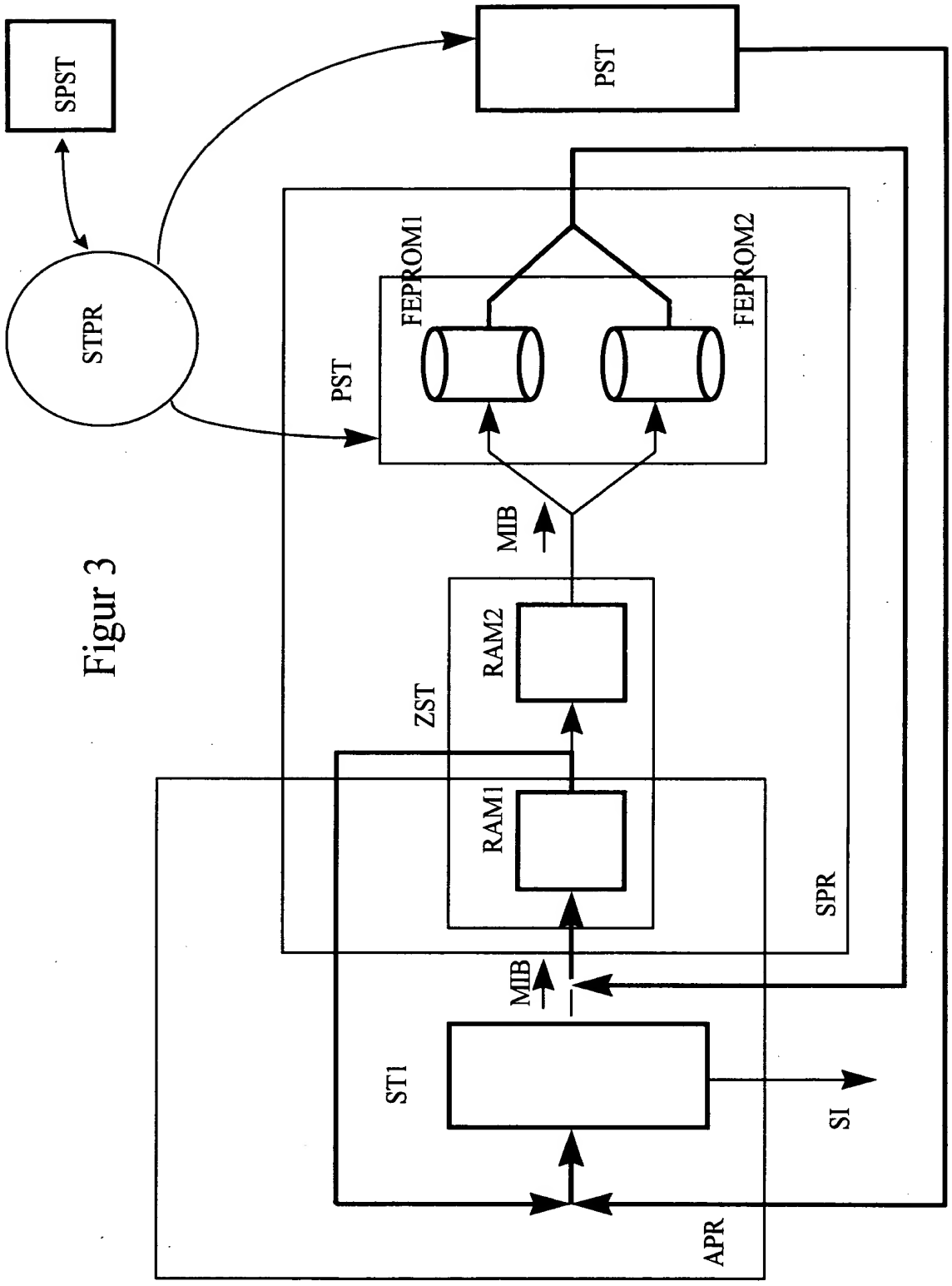
Figur 2

Figur 1



Figur 2




SPST
PST
STPR
PST
FEPROM1
FEPROM2
MIB
RAM2
ZST
RAM1
SPR
MIB
ST1
SI
APR

Figur 3

Figur 4